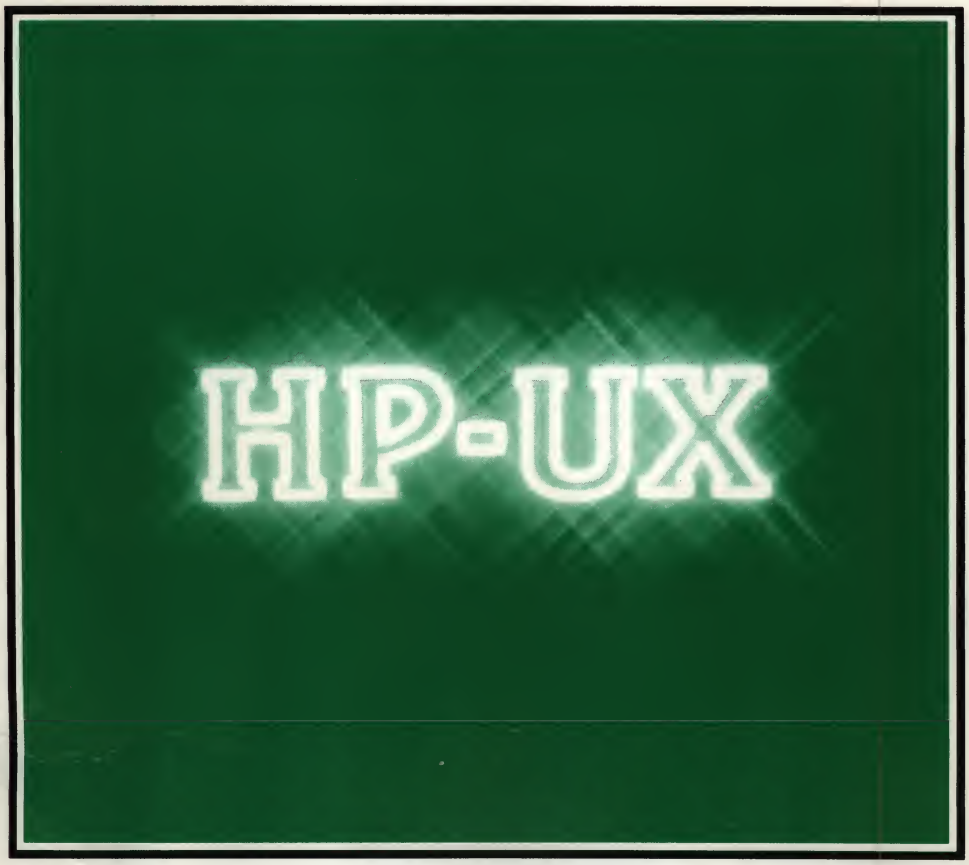
HP 9000 Series 300 Computers

HEWLETT PACKARD

HP 98550A
Software Portability Note



NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

HP 98550A
Software Portability Note

HP 9000 Series 300 Computers

HP Part Number 5959-1521

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive these updates or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

April 1987...Edition 1

Table of Contents

HP 98550A Software Portability Note

Introduction

The Starbase graphics library is Hewlett-Packard's programming library for access to HP graphics devices. A feature of this library is a set of procedures to do graphics transformations and output primitives using floating-point world coordinates. A current development effort at Technical Workstation Operation is to design and implement a corresponding set of **integer** world coordinate procedures, called the **integer interface**. This interface is intended to meet the needs of many 2-D applications in Electrical Engineering and some other CAD areas where coordinate data is naturally integer-based. This interface will not provide 3-D functionality.

This document is intended to provide an overview of the integer interface and aid in porting or writing applications to use it. The integer interface will be supported by an accelerator. This accelerator will enhance graphics performance on the HP 98550A display (as well as compatible displays that support the accelerator.) The interface will also be supported by Starbase for displays that do not have appropriate accelerator hardware.

Good graphics programming practices should be followed in any application that is expected to be portable to new devices and new display resolutions. Some of these practices are described in the chapters entitled "Device-independent Programming and Performance Enhancement" in the *Graphics Techniques HP-UX Concepts and Tutorials* manual. Other chapters in that manual describe basic graphics terminology and concepts not explained in this document.

Note

The procedure descriptions and other details of the integer interface shown in this document are **subject to change**. While most changes are expected to be minor, Hewlett-Packard reserves the right to alter any specification before final release.

Interface Overview

Integer interface is selected by placing the INT_XFORM flag in the **gopen** mode parameter. The default mode, INT_XFORM absent, is the floating-point interface.

The integer interface accepts coordinate data in 32-bit integer format. Internally, computations are done with 64-bit precision, and the results adjusted to 32 bits. Integer and floating-point calls cannot be mixed; if INT_XFORM is present, floating point calls are flagged as errors; if INT_XFORM is absent, integer calls are errors. The procedures appropriate for each interface are discussed later.

Transformation Pipelines

The integer interface provides either of two transformation pipelines. An integer transformation matrix stack is provided, similar to the floating-point matrix stack. The integer interface matrix representation differs from the floating-point matrix representation and is described later.

If MODEL_XFORM is present in the **gopen** mode parameter, user objects described in modeling coordinates can be individually rotated, scaled, and translated by a modeling transform as they are converted to world coordinates. World coordinates are in turn converted to virtual device coordinates by a viewing transform, and finally to device coordinates by the vdc-to-dc transform. The latter two stages are usually combined into a single viewing transformation. This architecture is useful when objects in a data base are to be manipulated and moved independently of each other.

If MODEL_XFORM is absent from the **gopen** mode parameter, separate control of the modeling and viewing transforms is not possible. Starbase recomputes a single transformation matrix whenever any part of the pipeline is changed. This architecture is slightly more efficient when elements of the data base do not need to be transformed in relation to each other, and when all the data must be quickly redrawn.

Since 3-D procedures are not provided in the integer interface, operations such as perspective divide or hither-yon clipping are not part of the integer pipeline.

Matrix Representation

Computations with 2-D transformations are done internally using 3x3 matrices. The integer interface defines matrix parameters as 3x2 integer arrays. The third column of each matrix can be assumed because it is always:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Scaling and translation are transformations easily represented by matrices with integer elements. Rotation depends on the sine and cosine functions and requires fractions to allow rotations that are not multiples of 90°.

In the Starbase integer interface, a fraction is represented as an integer mantissa multiplied by 2 raised to a negative power. The exponent is called the **radix**, and is expressed as a positive integer in the range 0 through 32. Each integer matrix has an associated radix value. Normally the radix is considered to apply to all elements of the matrix. However, if the *raw* parameter is set to TRUE in the matrix procedure call, the radix adjustment applies only to the rotation elements. This format allows the maximum range for translation while keeping good accuracy in rotation and scaling, and is used internally by Starbase. The rotation elements are mat[0][0], mat[0][1], mat[1][0], and mat[1][1]. The elements mat[2][0] and mat[2][1] are used for translation.

Converting a floating-point value to radix format can be done easily (the radix value being predetermined):

```
int_value = (int) (float_value * (1 << radix));
```

Conversely, to extract a floating point value from a radix representation:

```
float_value = ((float) int_value)/((float)(1 << radix));
```

The radix value can be thought of as the number of bits to the right of the binary point in the equivalent floating point arithmetic. When matrices are multiplied, the radix values must be added together. If the sum overflows the maximum value of 32, precision is lost as the matrix elements are adjusted to reduce the radix. This should only happen when several rotation and scaling matrices are concatenated, or when unusually high precision is specified.

Large integer values leave few bits remaining for fractional precision.

Procedure Descriptions

The integer interface procedures can be grouped into several categories:

- Output Primitives
- Primitive Attributes
- Transform Operations
- Input and Inquiry Routines
- Miscellaneous Other Routines
- Changes to Existing Routines

Each of these groups will be discussed below. Most changes simply substitute integer world coordinate parameters for the floating-point parameters in the equivalent Starbase procedures. You should consult the *Starbase Reference* manual to understand these changes. Only significant differences from existing routines will be discussed in the following sections.

The names of all procedures requiring integer world coordinate data or matrix parameters have been prefixed with **int** to distinguish them from floating-point and device-coordinate analogs.

Integer Interface Output Primitives

```
void intarc(fildes,radius,x_center,y_center,start,stop,close_type);
    int fildes,radius,x_center,y_center,close_type;
    float start,stop;

void intpartial_arc(fildes,radius,x_center,y_center,start,stop,close_type,
                    closure);
    int fildes,radius,x_center,y_center,close_type,closure;
    float start,stop;

void intcircle(fildes,radius,x_center,y_center);
    int fildes,radius,x_center,y_center;

void intpartial_circle(fildes,radius,x_center,y_center,closure);
    int fildes,radius,x_center,y_center,closure;

void intdraw2d(fildes,x,y);
    int fildes,x,y;

void intmove2d(fildes,x,y);
    int fildes,x,y;

void intpartial_polygon2d(fildes,clist,numverts,flags,closure);
    int fildes,numverts,flags,closure;
    int clist[];

void intpolygon2d(fildes,clist,numverts,flags);
    int fildes,numverts,flags;
    int clist[];

void intpolyline2d(fildes,clist,numpts,flags);
    int fildes,numpts,flags;
    int clist[];

void intpolymarker2d(fildes,clist,numpts,flags);
    int fildes,numpts,flags;
    int clist[];

void intrectangle(fildes,x1,y1,x2,y2);
    int fildes,x1,y1,x2,y2;
```

```
void inttext2d(fildes,x,y,string,xform);
    int fildes,x,y,xform;
    char *string;
```

Characters are drawn using the default text alignment for the current text path:

Path	Normal Horizontal	Normal Vertical
right	left	baseline
left	right	baseline
up	center	baseline
down	center	top

Only fonts 1 and 2 (see *text_font_index(3g)*) work correctly; others may not draw complete characters. All control characters in the string are ignored (except for space, octal 40).

Integer Interface Primitive Attributes

```
void intcharacter_height(fildes,height);
    int fildes,height;

void intcharacter_width(fildes,width);
    int fildes,width;

void intline_repeat_length(fildes,length);
    int fildes,length;

void intmarker_size(fildes,size,mode);
    int fildes,size,mode;

void intperimeter_repeat_length(fildes,length);
    int fildes,length;

void inttext_orientation2d(fildes,up_x,up_y,base_x,base_y);
    int fildes,up_x,up_y,base_x,base_y;
```

Integer Interface Transform Operations

```
void intclip_rectangle(fildes,lower_left_x,upper_right_x,lower_left_y,
                       upper_right_y);
    int fildes,lower_left_x,upper_right_x,lower_left_y,upper_right_y;
```

Defaults:

```
(lower_left_x,upper_right_x,lower_left_y,upper_right_y) = (0,32767,0,32767)
```

```
void intconcat_matrix2d(matrix1,matrix2,result,radix1,radix2,radix,raw);
    int matrix1[3][2],matrix2[3][2],result[3][2];
    int radix1,radix2,*radix,raw;

void intconcat_transformation2d(fildes,xform,radix,sequence,stack,raw);
    int fildes,radix,sequence,stack raw;
    int xform[3][2];

void intpop_matrix2d(fildes,xform,radix,raw);
    int fildes,*radix,raw;
    int xform[3][2];

void intpush_matrix2d(fildes,xform,radix,raw);
    int fildes,radix,raw;
    int xform[3][2];

void intreplace_matrix2d(fildes,xform,radix,raw);
    int fildes,radix,raw;
    int xform[3][2];

void inttransform_point2d(fildes,mode,inx,iny,outx,outy);
    int fildes,mode,inx,iny,*outx,*outy;

void intvdc_extent(fildes,xmin,ymin,xmax,ymax);
    int fildes,xmin,ymin,xmax,ymax;

void intview_matrix2d(fildes,xform,radix,usage,raw);
    int fildes,radix,usage,raw;
    int xform[3][2];

void intview_port(fildes,x1,y1,x2,y2);
    int fildes,x1,y1,x2,y2;

void intview_window(fildes,x1,y1,x2,y2);
    int fildes,x1,y1,x2,y2;
```

Integer Interface Input Routines and Inquiries

```
void intinquire_text_extent2d(fildes,string,xform,extent);
    int fildes,xform,extent[8];
    char *string;
```

On return, the extent array contains:

extent[0]:	concatenation point (x coordinate)
extent[1]:	concatenation point (y coordinate)
extent[2]:	lower left corner (x coordinate)
extent[3]:	lower left corner (y coordinate)
extent[4]:	upper left corner (x coordinate)
extent[5]:	upper left corner (y coordinate)
extent[6]:	upper right corner (x coordinate)
extent[7]:	upper right corner (y coordinate)

```
void intinq_pick_window(fildes,px_min,py_min,px_max,py_max);
    int fildes,*px_min,*py_min,*px_max,*py_max;

void intrequest_locator2d(fildes,ordinal,timeout,valid,x,y);
    int fildes,ordinal,*valid,*x,*y;
    float timeout;

void intsample_locator2d(fildes,ordinal,valid,x,y);
    int fildes,ordinal,*valid,*x,*y;
```

Miscellaneous Other Integer Interface Routines

```
void file_to_intbitmap(fildes,full_depth,spn,dpn,source,xstart,ystart,
                       update_cmap);
    int fildes,full_depth,spn,dpn;
    char *source;
    int xstart,ystart;
    int update_cmap;

void intbitmap_print(fildes,formatter,config,print_mode,full,xstart,ystart,
                     xlen,ylen,rotate,foreground,background,noback);
    int fildes;
    char *formatter,*config;
    int print_mode,full;
    int xstart,ystart;
    int xlen,ylen;
    int rotate,foreground,background,noback;
```

```
void intbitmap_to_file(fildes,full_depth,spn,dpn,dest,xstart,ystart,xlen,
                       ylen,store_cmap);
    int fildes,full_depth,spn,dpn;
    char *dest;
    int xstart,ystart;
    int xlen,ylen;
    int store_cmap;

void intblock_move(fildes,x_source,y_source length_x,length_y,x_dest,y_dest);
    int fildes,x_source,y_source,length_x,length_y,x_dest,y_dest);

void intblock_read(fildes,x_source,y_source,length_x,length_y,pixel_data,raw);
    int fildes,x_source,y_source,length_x,length_y;
    unsigned char *pixel_data;
    int raw;

void intblock_write(fildes,x_dest,y_dest,length_x,length_y, pixel_data,raw);
    int fildes,x_dest,y_dest,length_x,length_y;
    unsigned char *pixel_data;
    int raw;

void intecho_type2d(fildes,echo_number,echo_value,x,y);
    int fildes,echo_number,echo_value,x,y;

void intecho_update2d(fildes,echo_number,x,y);
    int fildes,echo_number,x,y;

void intset_pick_window(fildes,px_min,py_min,px_max,py_max);
    int fildes,px_min,py_min,px_max,py_max;
```

Changes to Existing Routines

```
void curve_resolution(fildes,coordinate_type,u_interior,v_interior,u_exterior,
                      v_exterior);
    int fildes,coordinate_type;
    float u_interior,v_interior,u_exterior,v_exterior;
```

This procedure now affects curves generated by `intarc`, `intpartial_arc`, `intcircle`, and `intpartial_circle`.

```
int gopen(path,kind,driver,mode);
char *path,*driver;
int kind,mode;
```

This procedure now accepts the INT_XFORM flag indicating that integer transforms are to be used

```
void set_hit_mode(fildes,hit_mode);
    int fildes,hit_mode;

void inquire_hit(fildes,hit);
    int fildes,*hit;
```

The picking aperture may now be set by `intset_pick_window`. Hits are registered for the integer interface output primitives listed above.

```
void pop_matrix(fildes);
    int fildes;
```

This procedure will pop a matrix off the integer matrix stack and discard it, just as in the floating point interface.

Programming with the Integer Interface

The integer interface is still under development; obviously it is not available in any released version of the Starbase library. If yours is an application that will benefit from the integer world coordinates and the potential performance accelerator, it is possible to adapt your code now to the interface and minimize the code effort required when the revised Starbase library is released.

The most modular approach to this problem is to implement a set of procedures that emulate the integer interface using the existing floating-point interface. If your data base is currently in integer format, your application is probably doing this now to use Starbase. This interface module can be linked into the application program until the Starbase library provides the defined procedures.

Most of the routines that must be implemented are simple "onion-skin" procedures that convert integer parameters to floating-point. This can be done easily in the C language using the cast operation; library and intrinsic functions are available in Pascal and FORTRAN77 to do the same functions.

Here is a simple example of an onion skin procedure for **intmarker_size**:

```
void intmarker_size (fildes,size,mode)
int fildes,size,mode;
{
 marker_size (fildes,(float)size,mode);
}
```

A more complex example that shows the conversion of a matrix from integer to floating-point format is **intpush_matrix2d**:

```
void intpush_matrix2d(fildes,xform,radix,raw)
int fildes,xform[3][2],radix,raw;
{
 float fxform[3][2],dev;

 if (radix == 0) {
  /* this optimization for radix == 0
     is not absolutely necessary */
  fxform[0][0] = xform[0][0];
  fxform[0][1] = xform[0][1];
  fxform[1][0] = xform[1][0];
  fxform[1][1] = xform[1][1];
  fxform[2][0] = xform[2][0];
  fxform[2][1] = xform[2][1];
 }
```

```
 else {
  /* the rotation elements must always be adjusted by the radix */
  dev = 1<<radix;
  fxform[0][0] = xform[0][0]/dev;
  fxform[0][1] = xform[0][1]/dev;
  fxform[1][0] = xform[1][0]/dev;
  fxform[1][1] = xform[1][1]/dev;
  /* the translation elements are adjusted if not raw format */
  if (raw) {
   fxform[2][0] = xform[2][0];
   fxform[2][1] = xform[2][1];
  }
  else {
   fxform[2][0] = xform[2][0]/dev;
   fxform[2][1] = xform[2][1]/dev;
  }
 }

 push_matrix2d(fildes,fxform);
}
```

The change to the **gopen** call can be made by defining a constant INT_XFORM as having value 0x20. However, postponing the change to the **gopen** call until the true integer interface is available may be just as easy as adding and then removing the constant definition.

Emulation procedures for all the integer interface routines described above compile to a relocatable file that can be linked into any Starbase application (preceding -lsb1 in the link sequence). If the code size is prohibitive, the procedures can be grouped into smaller relocatable modules collected into a library format by the *ar(1)* command. The linker can search such an archive and include only the needed relocatable files.

Programming Tips

Here are some other general tips on programming for the integer interface and for the accelerator:

1. Use large VDC extents. Small integer ranges will amplify the error due to integer arithmetic. The floating-point default range (0.0 to 1.0) is not useful in the integer interface.
2. Use integer coordinates for all graphics data. As stated above, mixing floating-point and integer calls is not possible, and converting data from one format to the other is error-prone and slow.
3. If you would prefer to convert from float to integer data some of the time, it may be valuable to write macros to do the conversion, so that the conversion can be modified or removed later.
4. Keep Starbase calls concentrated in a few files. This will make later modifications more localized.
5. Study the Starbase Graphics Techniques manual, especially the chapters about device-independent programming. Make sure all commitments to specific resolutions, numbers of planes, or other device features are made consciously. This is especially true of gescape operations that are not guaranteed to be supported in future drivers.
6. For raster operations, be aware of the effect of higher resolution devices that may be introduced in the future. Raster characters and menus will appear smaller on higher-resolution devices, unless care is taken to make the program adapt to the device resolution.
7. Follow performance hints provided in the driver chapters. These include advice about avoiding the alternation of attribute changes with output primitives, the relative costs of various drawing modes, and other driver-specific information.
8. Disjoint vectors should be placed into an array (with move/draw indicator if needed) and performed using the polyline procedures. Polylines are much more efficient than moves and draws.
9. Rectangles drawn using the rectangle procedure are somewhat faster than if done as unrotated polygons.
10. The accelerator is expected to provide hardware support for circles drawn with `intcircle`. Calls to `intarc`, `intpartial_arc`, and `intpartial_circle` will not perform as well, because the accelerator will not support them.

11. The accelerator will perform faster on non-overlapped windows than on overlapped windows.

Starbase Procedures and Coordinate Systems

Some Starbase procedures are only used with a single coordinate system, and some are common to more than one. A list of Starbase procedures and their related coordinate systems are shown below.

Floating Point Only

append_text	arc	backface_control
bitmap_to_file	bitmap_print	block_move
block_read	block_write	character_height
character_width	clip_depth	clip_rectangle
concat_matrix	concat_transformation2d	concat_transformation3d
dc_to_vdc	default_knots	define_trimming_curve
depth_cue	depth_indicator	draw2d
draw3d	echo_type	echo_update
ellipse	file_to_bitmap	hidden_surface
inquire_pick_depth	inquire_pick_window	inquire_text_extent
light_ambient	light_model	light_source
light_switch	line_repeat_length	marker_orientation
marker_size	move2d	move3d
partial_arc	partial_ellipse	partial_polygon2d
partial_polygon3d	perimeter_repeat_length	polygon2d
polygon3d	polyline2d	polyline3d
polymarker2d	polymarker3d	pop_matrix2d
pop_matrix3d	push_matrix2d	push_matrix3d
read_locator_event	rectangle	replace_matrix2d
replace_matrix3d	request_locator	sample_locator
set_locator	set_pick_depth	set_pick_window
shade_mode	shade_range	spline_curve

spline_curve2d
spline_curve3d
spline_surface
surface_model
text2d
text3d
text_alignment
text_line_path
text_line_space
text_orientation2d
text_orientation3d
transform_point
transform_points
u_knot_vector
v_knot_vector
vdc_extent
vdc_to_dc
vdc_to_wc
view_camera
view_matrix2d
view_matrix3d
view_port
view_volume
view_window
viewpoint
wc_to_vdc
zbuffer_switch

Integer Only

file_to_intbitmap
intarc
intbitmap_to_file
intbitmap_print
intblock_move
intblock_read
intblock_write
intcharacter_height
intcharacter_width
intcircle
intclip_rectangle
intconcat_matrix2d
intconcat_transform2d
intdraw2d
intecho_type2d
intecho_update2d
intinquire_pick_window
intinquire_text_extent2d
intline_repeat_length
intmarker_size
intmove2d
intpartial_arc
intpartial_circle
intpartial_polygon2d
intperimeter_repeat_length
intpolygon2d
intpolyline2d
intpolymarker2d
intpop_matrix2d
intpush_matrix2d
intrectangle
intreplace_matrix2d
intrequest_locator2d
intsample_locator2d
intset_pick_window
inttext2d
inttext_orientation2d
inttransform_point2d
intvdc_extent
intview_matrix2d
intview_port
intview_window

Device Coordinate Only

dcbitmap_to_file
dcbitmap_print
dcblock_move
dcblock_read
dcblock_write
dccharacter_height
dccharacter_width
dcdraw
dcecho_type
dcecho_update
dcmarker_size
dcmove
dcpartial_polygon
dcpolygon
dcpolyline
dcpolymarker
dcrectangle
dctext
file_to_dcbitmap

Common to Floating Point and Integer

character_expansion_factor
character_slant
clip_indicator
curve_resolution
flush_matrices
inquire_hit
intra_character_space
pop_matrix
push_vdc_matrix
set_hit_mode
set_p1_p2
text_path
text_precision
vdc_justification

Common to All Systems

await_event
await_retrace
background_color
background_color_index
bank_switch
buffer_mode
clear_control
clear_view_surface
dbuffer_switch
define_color_table
define_raster_echo
designate_character_set
disable_events
display_enable
double_buffer
drawing_mode
enable_events
file_print
fill_color
fill_color_index
fill_dither
gclose
gerr_message
gerr_print_control
gerr_procedure
gescape
gopen
initiate_request
inquire_choice
inquire_color_table
inquire_fb_configuration
inquire_gerror
inquire_id
inquire_input_capabilities
inquire_request_status
inquire_sizes
interior_style
line_color
line_color_index
line_type
make_picture_current
mapping_mode
marker_color
marker_color_index
marker_type
perimeter_color
perimeter_color_index
perimeter_type
read_choice_event
request_choice
sample_choice
set_signals
text_color
text_color_index
text_font_index
text_switching_mode
track
track_off
vertex_format
write_enable

HP Part Number
5959-1521
Printed in U.S.A. 4/87